

# **MÓDULO 6:**

## **POO CONCEPTOS CLAVE**

**Eduard Lara**

# INDICE

1. Operador this
2. Herencia
3. Polimorfismo
4. Clases abstractas y concretas
5. Métodos abstractos
6. Interficies
7. Métodos y clases finales
8. Referencia y clonación de objetos
9. Operador instanceof
10. Método \_destruct
11. Métodos estáticos

# 1. OPERADOR THIS

- L'operador "this" es una variable especial que **només té sentit dins del context d'un objecte instanciat**.
- Representa al mateix objecte, i serveix, per tant, per referir-se als mètodes i atributs d'ell mateix.

```
8  <?php
9  //Clase Padre
10 class Operacion {
11     //Atributos de clase
12     protected $valor1;
13     protected $valor2;
14     protected $resultado;
15     //Metodos de clase
16     public function cargar1($v)
17     {
18         $this->valor1=$v;
19     }
20     public function cargar2($v)
21     {
22         $this->valor2=$v;
23     }
24     public function imprimirResultado()
25     {
26         echo $this->resultado.'<br>';
27     }
28 }
```

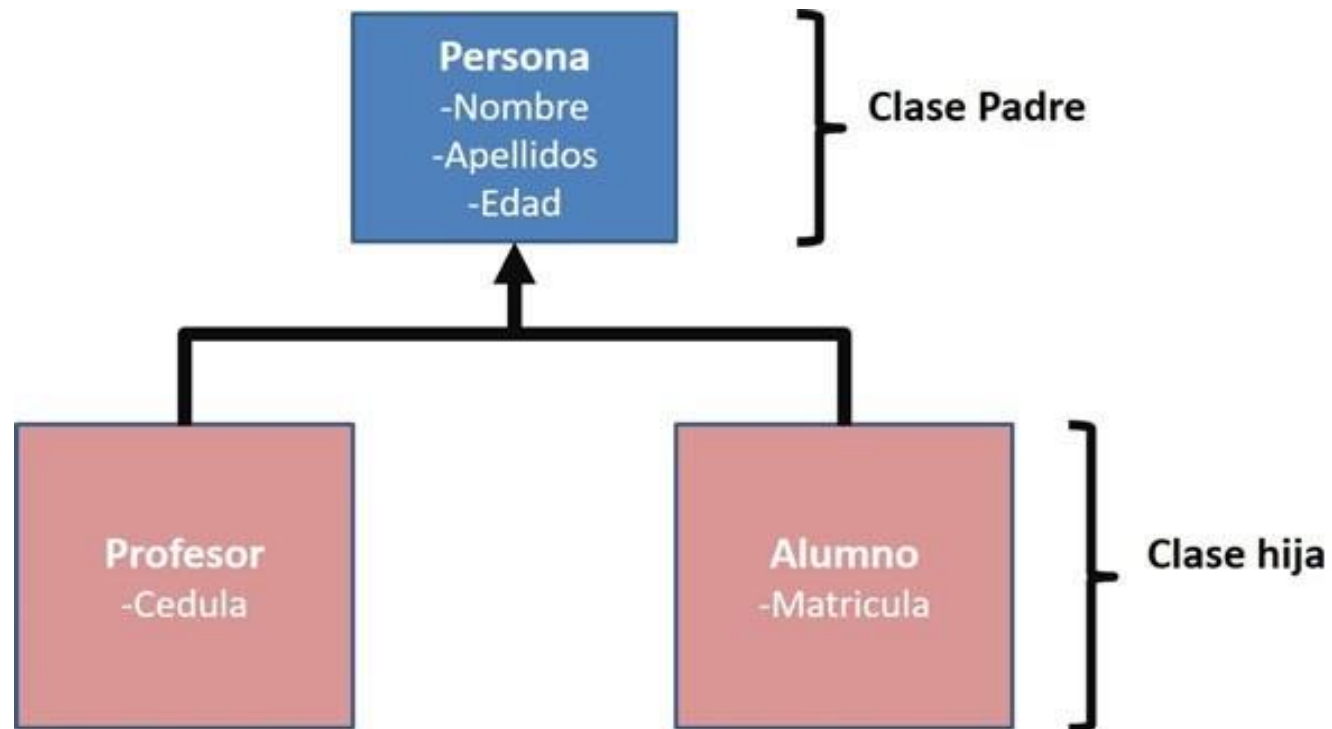
Aquí, per exemple,

**\$this**→**valor1** refereix a l'atribut de l'objecte, i li assigna el valor de la variable **\$v**.

Si la variable que entra pel paràmetre tingués el mateix nom(\$valor1) no hi hauria conflicte sempre i quan ens referíssim a l'atribut de l'objecte amb \$this.

## 2. HERENCIA

L'**herència** és un mecanisme de POO mitjançant el qual podem crear **jerarquies** de classes.



## 2. HERENCIA



## 2. HERENCIA

```
8  <?php
9  //Clase Padre
10 class Operacion {
11     //Atributos de clase
12     protected $valor1;
13     protected $valor2;
14     protected $resultado;
15     //Metodos de clase
16     public function cargar1($v)
17     {
18         $this->valor1=$v;
19     }
20     public function cargar2($v)
21     {
22         $this->valor2=$v;
23     }
24     public function imprimirResultado()
25     {
26         echo $this->resultado.'<br>';
27     }
28 }
```

La *keyword* **extends** ens serveix per a indicar de quina classe heretarem

```
29 //Clase hija
30 class Suma extends Operacion{
31     //Metodos de clase
32     public function operar()
33     {
34         $this->resultado=$this->valor1+$this->valor2;
35     }
36 }
37 //Clase hija
38 class Resta extends Operacion{
39     //Metodos de clase
40     public function operar()
41     {
42         $this->resultado=$this->valor1-$this->valor2;
43     }
44 }
```

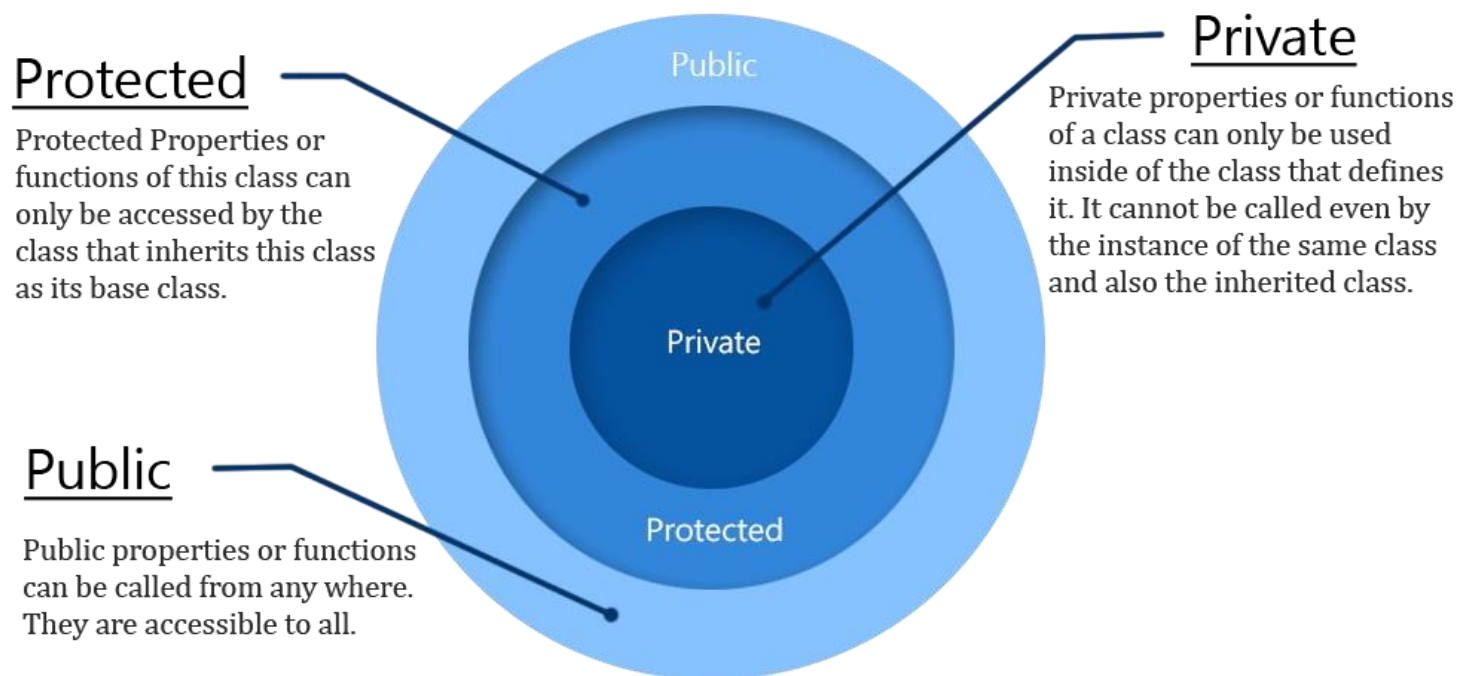
La principal “gràcia” d’aquest mecanisme es que s’hereten els mètodes i atributs depenent de la seva **visibilitat**(terme del qual parlem més endavant). En aquest cas, les classes Suma i Resta hereten \$valor1,\$valor2,\$valor3 i els mètodes cargar1,cargar2 e imprimirResultado, afegint-ho a les seves pròpies característiques.

## 2. HERENCIA

```
45 //Creacion e inicializacion de objetos
46 $suma=new Suma();
47 $suma->cargar1(10);
48 $suma->cargar2(10);
49 $suma->operar();
50 echo 'El resultado de la suma de 10+10 es: ';
51 $suma->imprimirResultado();
52 $resta=new Resta();
53 $resta->cargar1(10);
54 $resta->cargar2(5);
55 $resta->operar();
56 echo 'El resultado de la diferencia de 10-5 es: ';
57 $resta->imprimirResultado();
58
59 ?>
```

# 3. POLIMORFISMO

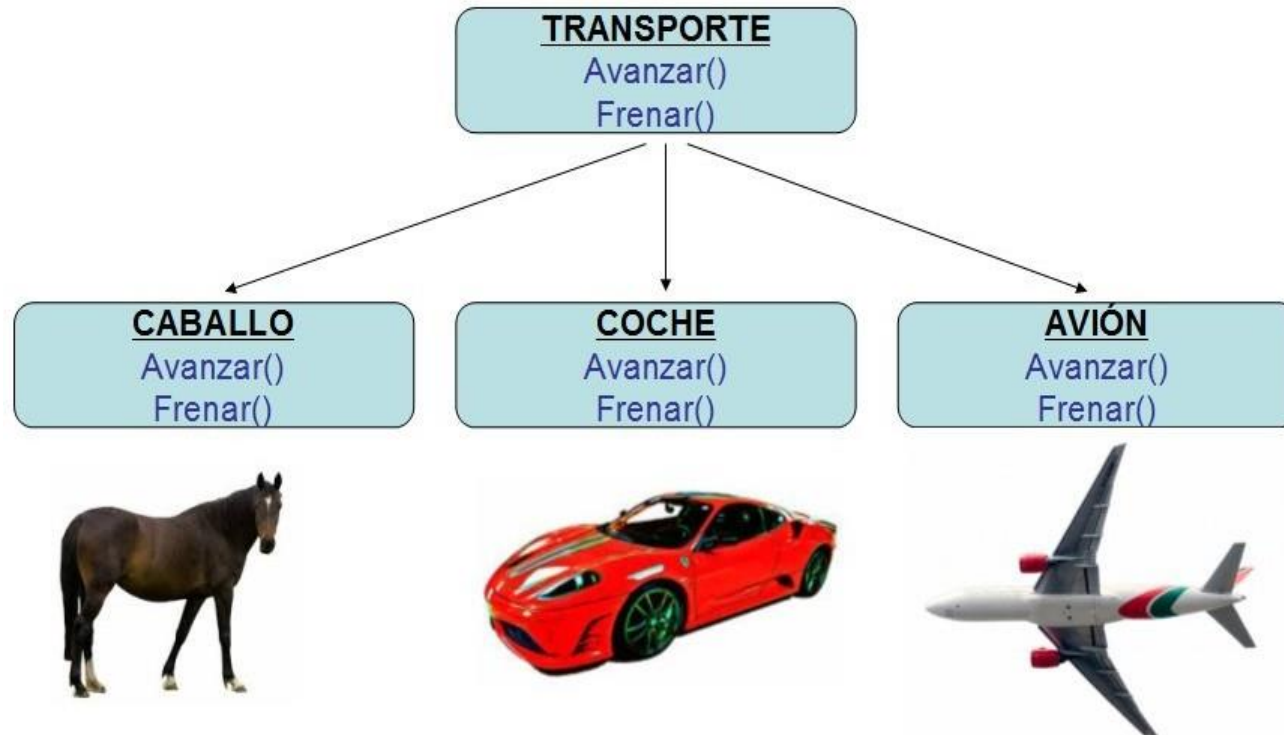
- La **visibilitat** és un mecanisme de POO que ens permet **limitar l'accés de mètodes o atributs d'una classe**.
- Les *keywords* **protected**, **private** o **public** davant d'un atribut o mètode d'una classe signifiquen, doncs:





### 3. POLIMORFISMO

El **polimorfisme** és un mecanisme de POO mitjançant el qual podem donar varies formes a un mateix objecte, permetent, entre d'altres coses, respondre de manera diferent a crides a funcions del mateix nom. El polimorfisme té sentit gràcies a l'**herència** en POO.



# 3. POLIMORFISMO

## Sobrescritura de metodos

Recuperem l'exemple anterior on la classe “pare” Operacion té com a “filles” les classes Suma y resta

```
117 <?php
118 //Clase Padre
119 class Operacion {
120     //Atributos de clase
121     protected $valor1;
122     protected $valor2;
123     protected $resultado;
124     //Metodos de clase
125     public function cargar1($v)
126     {
127         $this->valor1=$v;
128     }
129     public function cargar2($v)
130     {
131         $this->valor2=$v;
132     }
133     public function imprimirResultado()
134     {
135         echo $this->resultado.'<br>';
136     }
137 }
```

```
138 //Clase hija
139 class Suma extends Operacion{
140     //Metodos de clase
141     public function operar()
142     {
143         $this->resultado=$this->valor1+$this->valor2;
144     }
145     public function imprimirResultado()
146     {
147         echo "La suma de $this->valor1 y $this->valor2 es:";
148         parent::imprimirResultado();
149     }
150 }
```

```
151 //Clase hija
152 class Resta extends Operacion{
153     //Metodos de clase
154     public function operar()
155     {
156         $this->resultado=$this->valor1-$this->valor2;
157     }
158     public function imprimirResultado()
159     {
160         echo "La diferencia de $this->valor1 y $this->valor2 es:";
161         parent::imprimirResultado();
162     }
163 }
```

# 3. POLIMORFISMO

## Sobreescritura de metodos

```
164 //Creacion e inicializacion de objetos
165 $suma=new Suma();
166 $suma->cargar1(10);
167 $suma->cargar2(10);
168 $suma->operar();
169 $suma->imprimirResultado();
170 $resta=new Resta();
171 $resta->cargar1(10);
172 $resta->cargar2(5);
173 $resta->operar();
174 $resta->imprimirResultado();
175
176 ?>
```

“La suma de  
\$this→valor1 y  
\$this→valor2 es: ”

“La diferencia de  
\$this→valor1 y  
\$this→valor2 es: ”

Gràcies a la sobreescritura de mètodes que ens permet el polimorfisme, el mètode imprimirResultado es comportarà de manera diferent depenent del tipus d'objecte (suma o resta en aquest cas) que l'executi

# 3. POLIMORFISMO

## Sobreescritura de métodos (constructor)

També es poden sobreescriure mètodes per defecte de les classes com en **constructor**

```
187 //Clase Padre
188 class Operacion {
189     //Atributos de clase
190     protected $valor1;
191     protected $valor2;
192     protected $resultado;
193     //Constructor de clase
194     public function __construct($v1,$v2)
195     {
196         $this->valor1=$v1;
197         $this->valor2=$v2;
198     }
199     //Metodos de clase
200     public function imprimirResultado()
201     {
202         echo $this->resultado.'<br>';
203     }
204 }
```

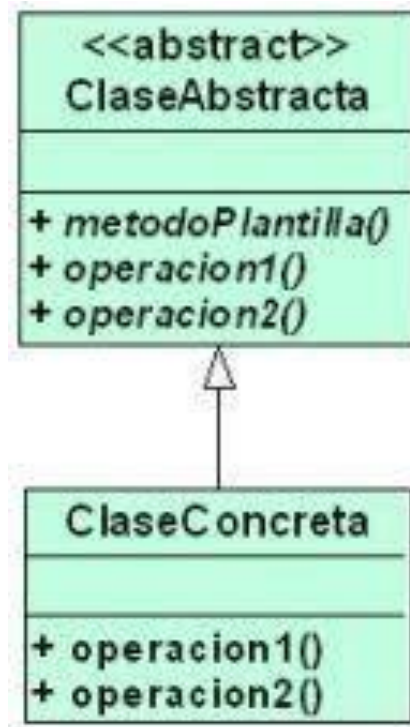
```
205 //Clase hija
206 class Suma extends Operacion{
207     //Atributos de clase
208     protected $titulo;
209     //Constructor de clase
210     public function __construct($v1,$v2,$tit)
211     {
212         Operacion::__construct($v1,$v2);
213         $this->titulo=$tit;
214     }
215     //Metodos de clase
216     public function operar()
217     {
218         echo $this->titulo;
219         echo $this->valor1.'+'. $this->valor2.' es ';
220         $this->resultado=$this->valor1+$this->valor2;
221     }
222 }
```

# 3. POLIMORFISMO

## Sobreescritura de métodos (constructor)

```
223 //Creacion e inicializacion de objetos
224 $suma=new Suma(10,10,'Suma de valores:');
225 $suma->operar();
226 $suma->imprimirResultado();
227
228 ?>
229 </body>
230 </html>
```

# 4. CLASES ABSTRACTAS Y CONCRETAS



Una classe abstracta té per objectiu agrupar atributs i mètodes que després seran heretats per altres subclasses.

**No** es pot instanciar

# 4. CLASES ABSTRACTAS Y CONCRETAS

```
238 <?php
239 //Clase abstracta padre
240 abstract class Operacion {
241     //Atributos de clase
242     protected $valor1;
243     protected $valor2;
244     protected $resultado;
245     //Metodos de clase
246     public function cargar1($v)
247     {
248         $this->valor1=$v;
249     }
250     public function cargar2($v)
251     {
252         $this->valor2=$v;
253     }
254     public function imprimirResultado()
255     {
256         echo $this->resultado.'<br>';
257     }
258 }
```

La *keyword* **abstract** ens serveix per programar una classe abstracta en PHP

# 4. CLASES ABSTRACTAS Y CONCRETAS

```
259 //Clase hija
260 class Suma extends Operacion{
261     //Metodos de clase
262     public function operar()
263     {
264         $this->resultado=$this->valor1+$this->valor2;
265     }
266 }
267 //Clase hija
268 class Resta extends Operacion{
269     //Metodos de clase
270     public function operar()
271     {
272         $this->resultado=$this->valor1-$this->valor2;
273     }
274 }
```

```
275 //Creacion e inicializacion de objetos
276 $suma=new Suma();
277 $suma->cargar1(10);
278 $suma->cargar2(10);
279 $suma->operar();
280 echo 'El resultado de la suma de 10+10 es: ';
281 $suma->imprimirResultado();
282 $resta=new Resta();
283 $resta->cargar1(10);
284 $resta->cargar2(5);
285 $resta->operar();
286 echo 'El resultado de la diferencia de 10-5 es: ';
287 $resta->imprimirResultado();
288
289 ?>
```

Que **Operacion** sigui una classe **abstracta** no ens evita que les classes que hereden d'ella funcionin com en exemples anteriors. Així doncs, aquí tindriem que **Suma** i **Resta** son **concrecions** de la classe **Operacion**



# 5. METODOS ABSTRACTOS

- Ara bé, si volem que les subclasses implementin **obligatoriament** determinats comportaments (mètodes) podem definir aquests **mètodes** com a **abstractes**.
- Un **mètode abstracte** es declara en una classe però **no s'implementa**.

```
public abstract function nombreFuncion();
```

# 5. METODOS ABSTRACTOS

*Keyword*  
**abstract**

```
299 <?php
300 //Clase abstracta padre
301 abstract class Operacion {
302     //Atributos de clase
303     protected $valor1;
304     protected $valor2;
305     protected $resultado;
306     //Metodos de clase
307     public function cargar1($v)
308     {
309         $this->valor1=$v;
310     }
311     public function cargar2($v)
312     {
313         $this->valor2=$v;
314     }
315     public function imprimirResultado()
316     {
317         echo $this->resultado.'<br>';
318     }
319     public abstract function operar();
320 }
```

# 5. METODOS ABSTRACTOS

```
321 //Clase hija
322 class Suma extends Operacion{
323     //Metodos de clase
324     public function operar()
325     {
326         $this->resultado=$this->valor1+$this->valor2;
327     }
328 }
329 //Clase hija
330 class Resta extends Operacion{
331     //Metodos de clase
332     public function operar()
333     {
334         $this->resultado=$this->valor1-$this->valor2;
335     }
336 }
```


```
337 //Creacion e inicializacion de objetos
338 $suma=new Suma();
339 $suma->cargar1(10);
340 $suma->cargar2(10);
341 $suma->operar();
342 echo 'El resultado de la suma de 10+10 es: ';
343 $suma->imprimirResultado();
344 $resta=new Resta();
345 $resta->cargar1(10);
346 $resta->cargar2(5);
347 $resta->operar();
348 echo 'El resultado de la diferencia de 10-5 es: ';
349 $resta->imprimirResultado();
350
351 -?>
```

Tot funcionarà exactament igual amb la diferència de que, en aquest cas, si no implementem a les classes Suma i Resta el mètode operar() es produirà un error

## 6. INTERFÍCIES

- Una **interfície** és una col·lecció de mètodes definits (no implementats) i que pot contenir també **valors constants**.
- Les interfícies més que heredarse, més aviat **s'implementen**, lo qual implicarà en la classe on sigui utilitzada, la **obligatòria implementació** dels mètodes definits a la interfície, **podent ser les implementacions diferents** entre **diferents classes**.

*Keyword* **interface** per  
a definir interfícies



```
interface Barco {  
    ...  
}
```

# 6. INTERFICIES

## Exemple Vaixell(Barco)

```
interface Barco { //una interfaz solo puede tener métodos públicos
    function hundirse();
    function atracar();
    function desembarcar();
}
```

Per implementar una interfície a una classe, ho podem fer tal que així.

```
class HidroAvion implements Barco {
```

# 6. INTERFICIES

## Exemple Vaixell(Barco)

```
class HidroAvion implements Barco, Avion {
```

- Una classe pot implementar més d'una interfície, separant-les amb comes.
- La idea de les **interfícies** és que podem tenir mètodes comuns a classes que en principi no tenen una relació de **pare-filla**. Es podria dir que és com si fossin classes cosines.
- Altra idea seria entendre les **interfícies** com a elements que “només” defineixen **comportaments**.

# 6. INTERFICIES

## Exemple Vaixell (Barco)

```
640 <?php
641 interface Barco { //una interfaz solo puede tener métodos públicos
642     function hundirse();
643     function atracar();
644     function desembarcar();
645 }
646 interface Avion {
647     function despegar();
648     function aterrizar();
649 }
650 class HidroAvion implements Barco, Avion {
651     public function aterrizar() {
652     }
653     public function atracar() {
654     }
655     public function desembarcar() {
656     }
657     public function despegar() {
658     }
659     public function hundirse() {
660     }
661 }
662 $hidro = new HidroAvion();
663 ?>
```

# 7. METODOS Y CLASES FINALES

Si a un mètode li afegim la *keyword* “final” significa que cap subclasse el pot sobreescriure. També podriem aplicar aquest modificador a una classe, indicant, llavors, que aquesta classe no es pot heretar.

```
361 <?php
362 //Clase padre
363 class Operacion {
364     //Atributos de clase
365     protected $valor1;
366     protected $valor2;
367     protected $resultado;
368     //Constructor de clase
369     public function __construct($v1,$v2)
370     {
371         $this->valor1=$v1;
372         $this->valor2=$v2;
373     }
374     //Metodos de clase
375     public final function imprimirResultado()
376     {
377         echo $this->resultado.'<br>';
378     }
379 }
```



# 7. METODOS Y CLASES FINALES

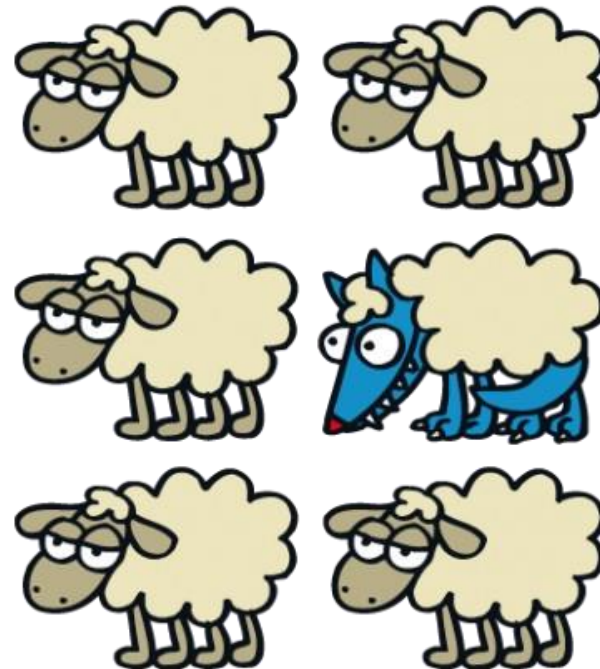
```
380 //Clase hija final
381 final class Suma extends Operacion{
382     //Atributos de clase
383     private $titulo;
384     //Constructor de clase
385     public function __construct($v1,$v2,$tit)
386     {
387         Operacion::__construct($v1,$v2);
388         $this->titulo=$tit;
389     }
390     //Metodos de clase
391     public function operar()
392     {
393         echo $this->titulo;
394         echo $this->valor1.'+'. $this->valor2.' es ';
395         $this->resultado=$this->valor1+$this->valor2;
396     }
397 }
```

# 7. METODOS Y CLASES FINALES

```
398 //Creacion e inicializacion de objetos
399 $suma=new Suma(10,10,'Suma de valores:');
400 $suma->operar();
401 $suma->imprimirResultado();
402
403 - ?>
```

## 8. REFERENCIA Y CLONACION DE OBJETOS

- Quan assignem una variable de tipus objecte a altra variable, lo que estem fent és guardar **referència de l'objecte**. **No s'està creant un nou objecte**, sino una altra variable mitjançant la qual, podriem accedir al mateix objecte.
- Si volem clonar un objecte idèntic hem d'utilitzar l'operador **clone**.



# 8. REFERENCIA Y CLONACION DE OBJETOS

```
413 <?php
414 //Definicion de Clase
415 class Persona {
416     //Atributos de clase
417     private $nombre;
418     private $edad;
419     //Metodos de clase
420     public function fijarNombreEdad($nom,$ed)
421     {
422         $this->nombre=$nom;
423         $this->edad=$ed;
424     }
425     public function retornarNombre()
426     {
427         return $this->nombre;
428     }
429     public function retornarEdad()
430     {
431         return $this->edad;
432     }
433 }
```

# 8. REFERENCIA Y CLONACION DE OBJETOS

```
434 //Creacion e inicializacion de objetos
435 $personal=new Persona();
436 $personal->fijarNombreEdad('Juan',20);
437 $x=$personal;
438 echo 'Datos de la persona ($personal):';
439 echo $personal->retornarNombre().' - '.$personal->retornarEdad().'\n';
440 echo 'Datos de la persona ($x):';
441 echo $personal->retornarNombre().' - '.$personal->retornarEdad().'\n';
442 $x->fijarNombreEdad('Ana',25);
443 echo 'Después de modificar los datos\n';
444 echo 'Datos de la persona ($personal):';
445 echo $personal->retornarNombre().' - '.$personal->retornarEdad().'\n';
446 echo 'Datos de la persona ($x):';
447 echo $personal->retornarNombre().' - '.$personal->retornarEdad().'\n';
448 $persona2=clone($personal);
449 $personal->fijarNombreEdad('Luis',50);
450 echo 'Después de modificar los datos de personal\n';
451 echo 'Datos de la persona ($personal):';
452 echo $personal->retornarNombre().' - '.$personal->retornarEdad().'\n';
453 echo 'Datos de la persona ($persona2):';
454 echo $persona2->retornarNombre().' - '.$persona2->retornarEdad().'\n';
455
456 ?>
```

## Función Clone()

- PHP ens permet crear un mètode que es cridarà quan executem l'operador **clone**. Aquest mètode pot entre altres coses, inicialitzar alguns atributs.
- **Si no es defineix el mètode `__clone`, es farà una còpia idèntica de l'objecte que li passem com a paràmetre a l'operador clone.**

```
public function __clone()  
{  
    $this->atributo=0;  
}
```

# 8. REFERENCIA Y CLONACION DE OBJETOS

## Función Clone()

```
466 <?php
467 //Definicion de Clase
468 class Persona {
469     //Atributos de clase
470     private $nombre;
471     private $edad;
472     //Metodos de clase
473     public function fijarNombreEdad($nom,$ed)
474     {
475         $this->nombre=$nom;
476         $this->edad=$ed;
477     }
478     public function retornarNombre()
479     {
480         return $this->nombre;
481     }
482     public function retornarEdad()
483     {
484         return $this->edad;
485     }
486     public function __clone()
487     {
488         $this->edad=0;
489     }
490 }
```

# 8. REFERENCIA Y CLONACION DE OBJETOS

## Función Clone()

```
491 //Creacion e inicializacion de objetos
492 $personal=new Persona();
493 $personal->fijarNombreEdad('Juan',20);
494 echo 'Datos de $personal: ';
495 echo $personal->retornarNombre(). ' - ' . $personal->retornarEdad(). '<br>';
496 $persona2=clone($personal);
497 echo 'Datos de $persona2: ';
498 echo $persona2->retornarNombre(). ' - ' . $persona2->retornarEdad(). '<br>';
499
500 ->
```



## 9. OPERADOR INSTANCEOF

Quan tenim una llista d'objectes de diferents tipus i volem saber **si un objecte és d'una determinada** classe, a PHP tenim l'operador **instanceof**.

```
if ($vec instanceof Gerente)
    echo 'Los objetos son del mismo tipo';
}
```

## 9. OPERADOR INSTANCEOF

```
510 <?php
511 //Clase abstracta padre
512 abstract class Trabajador {
513     //Atributos de clase
514     protected $nombre;
515     protected $sueldo;
516     //Constructor de clase
517     public function __construct($nom,$sue)
518     {
519         $this->nombre=$nom;
520         $this->sueldo=$sue;
521     }
522     //Metodos de clase
523     public function retornarSueldo()
524     {
525         return $this->sueldo;
526     }
527 }
```

# 9. OPERADOR INSTANCEOF

```
528 //Clase hija
529 -class Empleado extends Trabajador {
530 -}
531 //Clase hija
532 -class Gerente extends Trabajador {
533 -}
```

```
534 //Creacion e inicializacion de objetos
535 $vec[]=new Empleado('juan',1200);
536 $vec[]=new Empleado('ana',1000);
537 $vec[]=new Empleado('carlos',1000);
538
539 $vec[]=new Gerente('jorge',25000);
540 $vec[]=new Gerente('marcos',8000);
541
542 $sumal=0;
543 $suma2=0;
544 for($f=0;$f<count($vec);$f++)
545 -{
546     if ($vec[$f] instanceof Empleado)
547         $sumal=$sumal+$vec[$f]->retornarSueldo();
548     else
549         if ($vec[$f] instanceof Gerente)
550             $suma2=$suma2+$vec[$f]->retornarSueldo();
551 -}
552 echo 'Gastos en sueldos de Empleados:'. $sumal.'  
';
553 echo 'Gastos en sueldos de Gerentes:'. $suma2.'  
';
```

# 10. METODO \_DESTRUCT

- ☐ El seu objectiu principal és alliberar recursos que l'objecte va sol·licitar com per exemple: connexió a bases de dades, creació d'imatges dinàmiques...
- ☐ És l'últim mètode que s'executa de la classe.
- ☐ S'executa de manera automàtica, és a dir, que no l'hem de cridar.
- ☐ S'ha d'anomenar \_destruct.
- ☐ No retorna dades.

# 10. METODO \_DESTRUCT

```
560 <?php
561 /////Definicion de Clase
562 class Banner {
563     //Atributos de clase
564     private $ancho;
565     private $alto;
566     private $mensaje;
567     private $imagen;
568     private $colorTexto;
569     private $colorFondo;
570     //Constructor de clase
571     public function __construct($an,$al,$men)
572     {
573         $this->ancho=$an;
574         $this->alto=$al;
575         $this->mensaje=$men;
576         $this->imagen=imageCreate($this->ancho,$this->alto);
577         $this->colorTexto=imageColorAllocate($this->imagen,255,255,0);
578         $this->colorFondo=imageColorAllocate($this->imagen,255,0,0);
579         imageFill($this->imagen,0,0,$this->colorFondo);
580     }
```

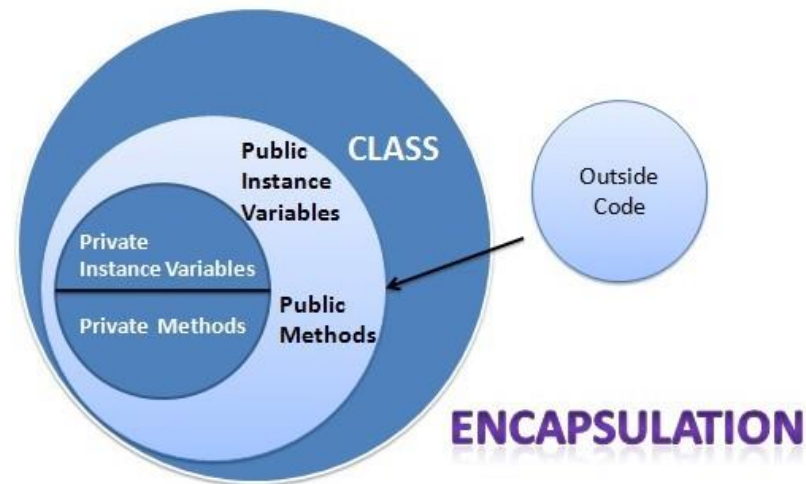
# 10. METODO \_DESTRUCT

```
581 //Metodos de clase
582 public function graficar()
583 {
584     imageString ($this->imagen,5,50,10, $this->mensaje,$this->colorFuente);
585     header ("Content-type: image/png");
586     imagePNG ($this->imagen);
587 }
588 public function __destruct()
589 {
590     imageDestroy($this->imagen);
591 }
592 }
```

```
593 //Creacion e inicializacion de objetos
594 $baner1=new Banner(428,45,'Sistema de Ventas por Mayor y Menor');
595 $baner1->graficar();
596
597 ?>
```

# 11. METODOS ESTATICOS

- Un mètode estàtic pertany a la classe però no pot accedir als atributs d'una instància.
- La característica fonamental és que un mètode estàtic es pot cridar sense haver d'instanciar la classe.
- Un mètode estàtic és lo més semblant a una funció de llenguatge estructurat, només que, s'encapsula a dins d'una classe.



# 11. METODOS ESTATICOS

```
607 <?php
608 //Definicion de Clase
609 class Cadena {
610     //Metodos de clase
611     public static function largo($cad)
612     {
613         return strlen($cad);
614     }
615     public static function mayusculas($cad)
616     {
617         return strtoupper($cad);
618     }
619     public static function minusculas($cad)
620     {
621         return strtolower($cad);
622     }
623 }
```



# 11. METODOS ESTATICOS

```
624 //Creacion e inicializacion de objetos
625 $c='Hola';
626 echo 'Cadena original:'. $c;
627 echo '<br>';
628 echo 'Largo:'.Cadena::largo($c);
629 echo '<br>';
630 echo 'Toda en mayúsculas:'.Cadena::mayusculas($c);
631 echo '<br>';
632 echo 'Toda en minúsculas:'.Cadena::minusculas($c);
633
634 -?>
```